Authenticating service-to-service

If your architecture is using multiple services, these services will likely need to communicate with each other.

You can use synchronous or asynchronous service-to-service communication:

For *asynchronous* communication, use

- Cloud Tasks (https://cloud.google.com/tasks/docs/creating-http-target-tasks#http_target_tasks_with_authentication_tokens) for one to one asynchronous communication
- Pub/Sub (https://cloud.google.com/run/docs/events/pubsub-push) for one to many asynchronous communication

For *synchronous* communication, one service invokes another one over HTTP using its endpoint URL. In this use case, it's a good idea to ensure that each service is only able to make requests to specific services. For instance, if you have a `login` service, it should be able to access the `user-profiles` service, but it probably shouldn't be able to access the `search` service.

First, you'll need to configure the receiving service to accept requests from the calling service:

1. Grant the Cloud Run Invoker (`roles/run.invoker`) role to the calling service identity (https://cloud.google.com/run/docs/securing/service-identity) on the receiving service. By default, this identity is `PROJECT_NUMBER-compute@developer.gserviceaccount.com`.

CONSOLE UI GCLOUD

1. Go to the Google Cloud Console:

   GO TO GOOGLE CLOUD CONSOLE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/RUN/)

2. Select the receiving service.
3. Click **Show Info Panel** in the top right corner to show the **Permissions** tab.

4. In the **Add members** field, enter the identity of the calling service.
5. Select the `Cloud Run Invoker` role from the **Select a role** drop-down menu.
6. Click **Add**.

In the calling service, you'll need to:

1. Create a Google-signed OAuth ID token with the audience (`aud`) set to the URL of the receiving service. This value must contain the schema prefix (`http://` or `https://`) and custom domains are currently not supported for the `aud` value.
2. Include the ID token in an `Authorization: Bearer ID_TOKEN` header in the request to the service.

★ **Note:** ID tokens are JSON Web Tokens (JWTs) (https://en.wikipedia.org/wiki/JSON_Web_Token) that expire approximately an hour after creation. If you fetch tokens from the metadata server (https://cloud.google.com/run/docs/securing/service-identity#identity_tokens), you will always get a valid token. If you choose to cache tokens yourself, you can decode the token and check the `exp` time see if you need to refresh the token.

NODEJS PYTHON GO

```
// Make sure to `npm install --save request-promise` or add the dependency to your
const request = require('request-promise');

const receivingServiceURL = ...

// Set up metadata server request
// See https://cloud.google.com/compute/docs/instances/verifying-instance-identity#
const metadataServerTokenURL = 'http://metadata/computeMetadata/v1/instance/service
const tokenRequestOptions = {
    uri: metadataServerTokenURL + receivingServiceURL,
    headers: {
        'Metadata-Flavor': 'Google'
    }
};

// Fetch the token, then provide the token in the request to the receiving service
request(tokenRequestOptions)
  .then((token) => {
    return request(receivingServiceURL).auth(null, null, true, token)
  })
  .then((response) => {
    res.status(200).send(response);
  })
  .catch((error) => {
    res.status(400).send(error);
  });
```

Calling from outside GCP

If you're invoking a service from a compute instance that doesn't have access to compute metadata (e.g. your own server), you'll have to manually generate the proper token:

1. Self-sign a service account JWT with the `target_audience` claim set to the URL of the receiving service.
2. Exchange the self-signed JWT for a Google-signed ID token, which should have the `aud` claim set to the above URL.
3. Include the ID token in an `Authorization: Bearer ID_TOKEN` header in the request to the service.

The Cloud IAP docs have sample code (https://cloud.google.com/iap/docs/authentication-howto#authenticating_from_a_service_account) to demonstrate this functionality.